The Python Papers Monograph 2: 18
Proceedings of PyCon Asia-Pacific 2010

# VPython Application To The Computer-aided Drug Design Problem

**George Loo**
Director, R & D
Asia Radio 2000 Pte Ltd
gloosg@yahoo.com.sg

**Ching Jianhong**
Ph. D. Candidate
Department of Pharmacy
Faculty of Science
National University of Singapore
chingjianhong@hotmail.com

## Abstract

Quantitative structure-activity relationship (QSAR) is the study of the mathematical relationship between the chemical and geometrical characteristics structure of a lead compound with its biological activity. The structures of the lead compounds and their receptors could be represented with 3D computer models. VPython is a package designed with the ease of making and manipulating such models. Researchers can then modify such drug molecular models for better biological activity in silico before synthesizing the drug. The value of using this strategy in drug discovery is to reduce costs and time in the prediction of drug activity.

## 1. Introduction

VPython (Scherer et al. 2000) is a 3D graphics library module added to the Python programming language. The module is called Visual. Users can create objects such as cylinders and cubes in a 3D space. VPython handles in a separate thread all the 3D rendering and navigation in this 3D window space, freeing the user to focus on the simulation and mathematical aspects of the problem they wish to solve or illustrate.

David Scherer, while an undergraduate at Carnegie Mellon University, started VPython in 2000 as a 2D package. Since then, with contributions from David Andersen, Ruth Chabay, Ari Heitner, Ian Peters, and Bruce Sherwood, VPython has grown to encompass 3D graphics as well. The latest version of VPython is 5.13, running in Python 2.6. A version of IDLE Python program editor called VIDLE is bundled with VPython. Programs described will be executed from VIDLE.

To illustrate the simplicity of using VPython, consider the 2 lines of code from file helloVPython.py:

```
from visual import *
sphere()
```


Figure. 1. The simplest VPython program

From these 2 lines of code, the user gets a window with a sphere in the centre. Using the right button on the mouse, the user can rotate the scene. All lighting and texturing of the sphere is handled by VPython. Pressing both the left and right mouse buttons together makes zooming of the scene possible.

The following are the 3D primitives in Visual 5 (Scherer 2009) :

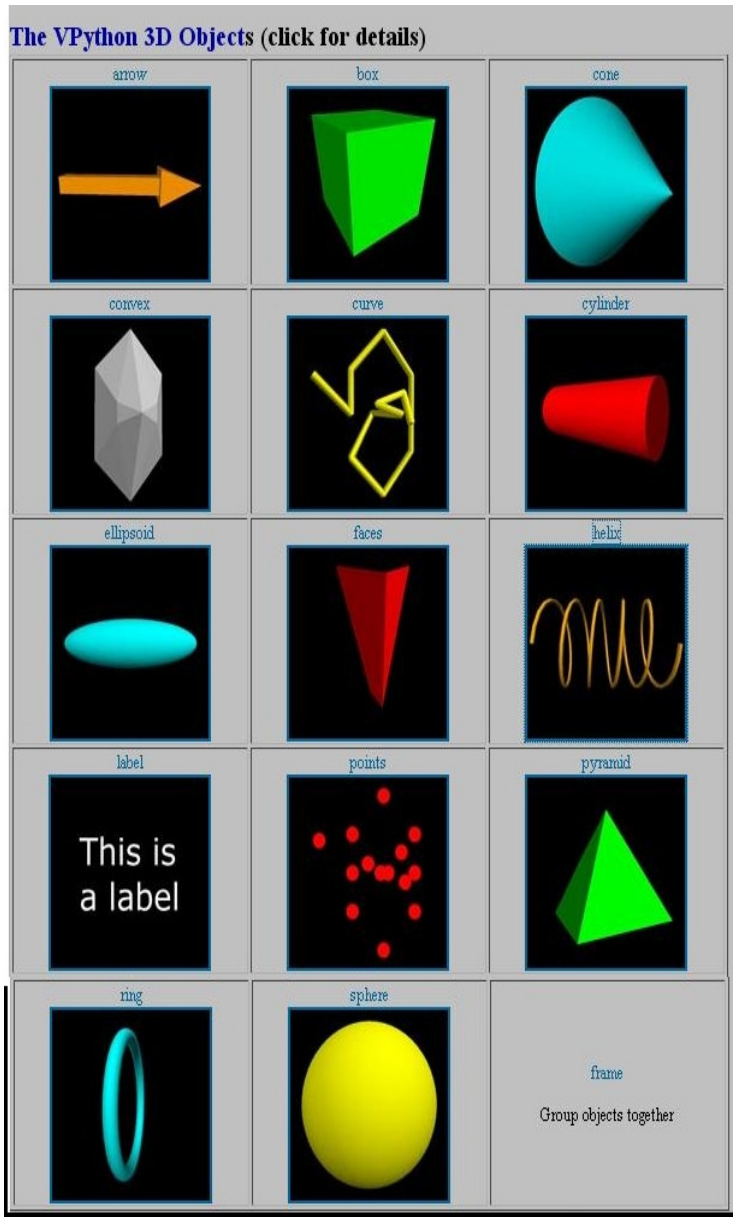



Figure 2. The 3D Objects in VPython

In addition, there are several mathematical libraries included in VPython:

1. NUMPY for numerical arrays
2. Math module from Python language
3. Vector operations - Magnitude, dot and cross product, rotation, etc.
4. Graphing - for illustrating the relations of variables
5. Factorial and Combinatorial functions - Special functions used in probability calculations

### 1.1 Examples in the VPYthon package

One of the examples in the VPython package illustrates how spheres and boxes show the interaction between gravity and a ball falling - bounce.py.

```
from visual import *
scene.title = "ball bounce"
floor = box(length=4, height=0.5, width=4, color=color.blue)
ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity = vector(0,-1,0)
dt = 0.01
while 1:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
```

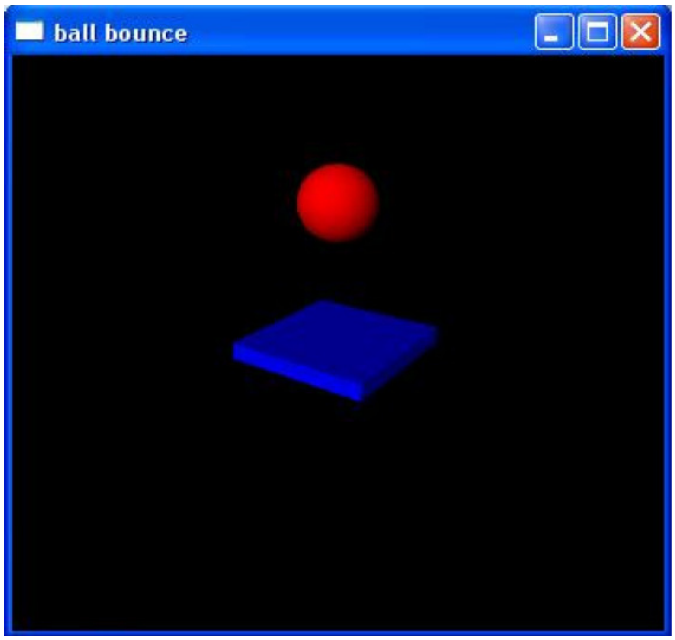



Figure 3. Ball bouncing example code

Of course, this ideal example does not take into account energy losses. The ball will bounce indefinitely.

VPython has mouse interaction features as well. The programmer can drag objects in the 3D space with the mouse. This is most useful for designing and manipulating simulated drug molecules.

An example of mouse interaction (dragging.py):

```
from visual import *
scene.range = 5 # fixed size, no autoscaling
ball = sphere(pos=(-3,0,0), color=color.cyan)
cube = box(pos=(+3,0,0), size=(2,2,2), color=color.red)
pick = None # no object picked out of the scene yet

while True:
    if scene.mouse.events:
        m1 = scene.mouse.getevent() # get event
        if m1.drag and m1.pick == ball: # if touched ball
            drag_pos = m1.pickpos # where on the ball
            pick = m1.pick # pick now true (not None)
        if m1.drag and m1.pick == cube: # if touched cube
            drag_pos = m1.pickpos # where on the cube
            pick = m1.pick # pick now true (not None)
        elif m1.drop: # released at end of drag
            pick = None # end dragging (None is false)
    if pick:
        # project onto xy plane, even if scene rotated:
```

```
new_pos = scene.mouse.project(normal=(0,0,1))
#new_pos = scene.mouse.pos
if new_pos != drag_pos: # if mouse has moved
    # offset for where the ball was clicked:
    pick.pos += new_pos - drag_pos
    drag_pos = new_pos # update drag position
```

There are limitations to the code above as there is no fine control and feedback to the user as to where the object has moved to. We will present our own code to address this problem.

An example included with VPython, gas.py, demonstrates the speed of the graphics in calculating the movements of a hundred gas atoms as well as their collisions and resulting vectors. In addition, a separate window displays the graph of accumulated collisions. It also illustrates the use of numeric arrays found in the included numpy package.

Numpy usage in gas.py (line 91):

```
r = pos-pos[:,newaxis] # all pairs of atom-to-atom vectors
rmag = sqrt(add.reduce(r*r,-1)) # atom-to-atom scalar distances
hit = less_equal(rmag,radius+radius[:,None])-identity(Natoms)
hitlist = sort(nonzero(hit.flat)[0]).tolist() # i,j encoded as i*Natoms+j
```

Collision detection in gas.py (line 119):

```
deltat = (-b+sqrt(d))/(2.*a) # t-deltat is when they made contact
pos[i] = pos[i]-(p[i]/mi)*deltat # back up to contact configuration
pos[j] = pos[j]-(p[j]/mj)*deltat
```

Histogram graphing facility in VPython (Gas.py, line 37):

```
observation = ghistogram(bins=arange(0.,3000.,deltav),
                         accumulate=1, average=1, color=color.red)
```

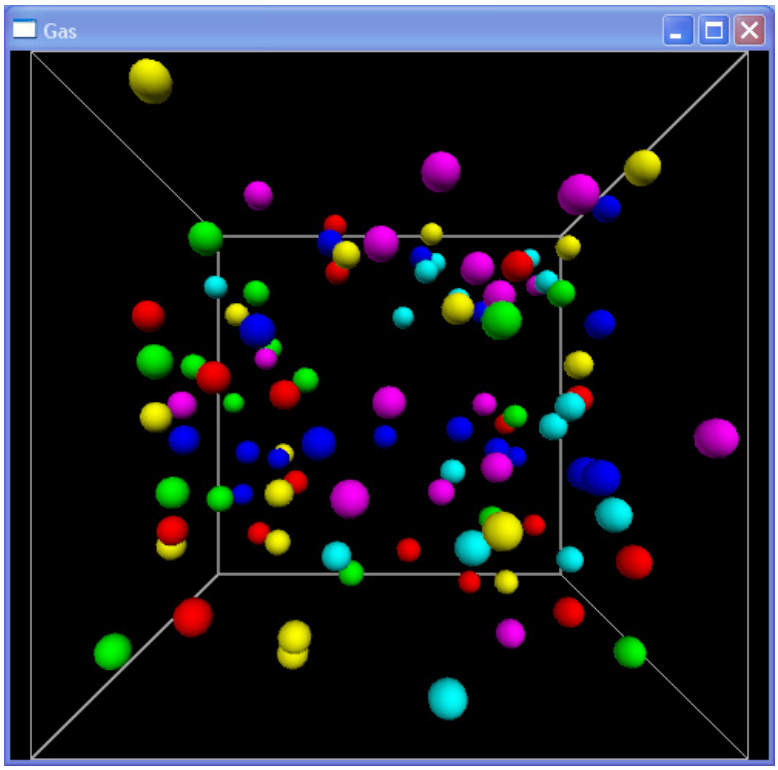



Figure 4. Gas.py: Tracking a hundred atoms in motion

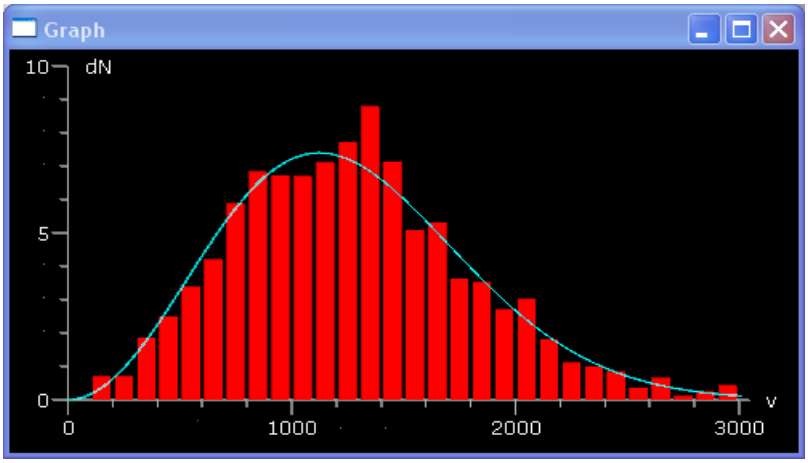



Figure 5. Gas.py: Graph of accumulated collisions

### 1.2 Uses in the education field

Numerous physics teachers have used VPython to teach their courses. Rob Salgado (2009) uses VPython to teach Doppler and Magnetic waves in his courses.

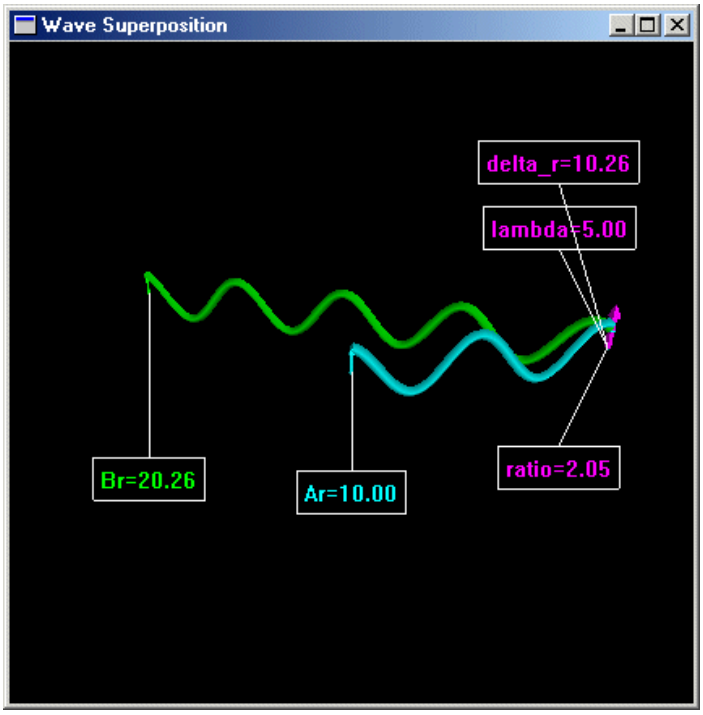



Figure 6. Wave Superposition using VPython (Salgado 2009)

Erik Thompson (2009) has made videos to teach physics using VPython.

The late Arthur Siegel (2006) used VPython to create dynamic geometric constructions

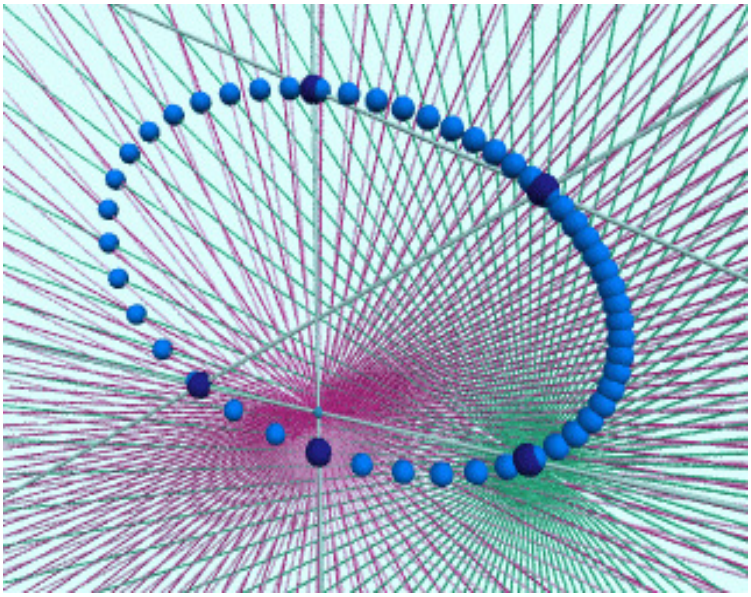


Figure 7. The Conic determined by 5 co-planar points by Siegel (2006)

Lensyl Urbano (2005) uses VPython to teach Science.

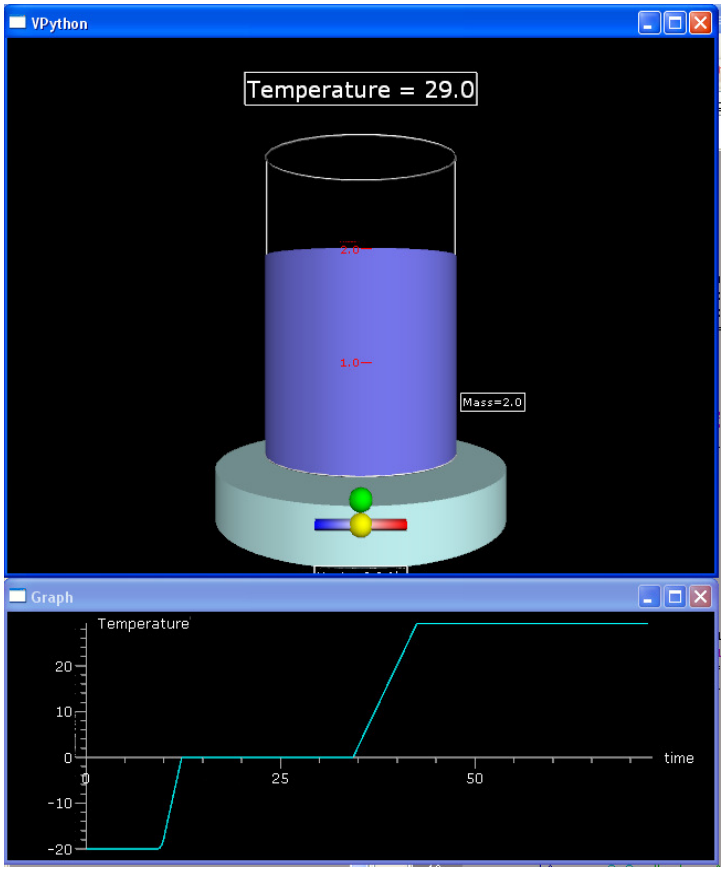



Figure 8. Water phase changes by Urbano (2005)

## 2. QSAR and the drug design problem

QSAR can be defined as the study of the mathematical relationship between the chemical and geometrical characteristics structure of a lead compound with its biological activity. Biological activity, in this case, refers to a particular activity being studied on biological systems such as the whole body or isolated cells (for eg. anticancer acitivity, antiviral, antibacterial, antihypertensive activity). Since the early works of Hansch, much advancement has been made in the field of QSAR. This technique has been gaining importance in the field of drug design as it enables high throughput screening of compounds for drugs. QSAR can be used as a complementary to combinatorial chemistry to virtually filter and screen large libraries of compounds. By enabling prediction of biological activity, QSAR can help reduce time and costs to screen each and every one of the compounds available by eliminating compounds with poor predicted activities or those that could potentially be toxic. In this case, the number of experiments that need to be conducted can be reduced and more focus can be put on low throughput experiments (Dudek 2006).

QSAR's most general mathematical form is:

Biological activity = $C_0+(C_1*P1)+...+(C_n*Pn)$,
Where C is the coefficient to be explained by QSAR and P is the descriptor of the molecule.

An application of QSAR to predict biological activities based on the molcule's hydrophobicity and sigma value is given below:

$\text{Log } (1/C)= k_1\pi+ k_2\sigma+ k_3,$

Where C is the concentration of the drug, $\pi$ is the hydrophobicity (how insoluble a substance is in water, measured by the partition coefficient of the substance between octanol and water) and $\sigma$ is the sigma value, indicating the electronic effects of the molecule's constituents. The sigma values are experimental values assigned to substituents in the chemical group, which is calculated from its resonance and inductive effects. A positive sigma value indicates how electron withdrawing the chemical group is, while a negative sigma value indicates how electron donating the chemical group is.

A drug is any substance that, when introduced into the body of a living organism, alters the normal or disease functions. Drug design is based on the observations of the reaction of a body to a test

compound. Compounds consist of two or more chemical elements (Brown et al. 2009). QSAR methodology tabulates such data for each of the elements and uses data mining techniques to attempt to discover a trend in the data. The goal is to be able to predict the effect of a certain combination of elements on a living organism.

## 2.1 Drug Design Process

### 2.1.1 Discovery

The initial step of drug discovery involves the identification of new active compounds, often called "hits", which are typically found by screening many compounds for the desired biological properties. These hits can come from natural sources, such as plants, animals, or fungi. More often, the hits can come from synthetic sources, such as stored compound collections and combinatorial chemistry.

Recent developments in robotics and miniaturization have incredibly accelerated and automated the screening process. Typically, a company will assay over 100,000 individual compounds using a method called high-throughput screening, before moving to the optimization step (Farlax 2010).

### 2.1.2 Optimization

The second step of drug discovery involves the modification of the hits in order to improve the biological properties of the compound by changing its pharmacophore. Using QSAR to modify lead compounds would be less tedious then having to physically synthesize the compounds. Moreover, such in silico methods could theoretically help to modify the compounds to exhibit the most potency, most selectivity, best pharmacokinetics and least toxicity. QSAR involves mainly physical chemistry and molecular docking tools, that lead to tabulated data and first and second order equations. There are many theories, being the most relevant Hansch's analysis that involves Hammett electronic parameters, Esteric parameters and logP parameters (Farlax 2010).

### 2.1.3 Development

Development of the lead compound involves thorough testing in vitro and in animal systems (in vivo). The final step involves rendering the lead compounds suitable for use in clinical trials. This involves the optimization of the drug for bulk production, and the preparation of suitable drug formulations (Farlax 2010).

## 2.2 Receptor Theory

A receptor (Selassie 2003), in the biochemistry context, is a/are protein molecule(s), found in either the plasma membrane or the cytoplasm of a cell, to which one or more specific kinds of signalling molecules may attach. A molecule which attaches to a receptor is called a ligand, and may be a peptide or other small molecule, such as a neurotransmitter, a hormone, a pharmaceutical drug, or a toxin. Each kind of receptor can bind only certain ligand shapes. Each cell typically has many receptors, of many different kinds.

An agonist is a drug that binds to a receptor of a cell and triggers a response by the cell. An agonist often mimics the action of a naturally occurring substance. An agonist produces an action. An antagonist blocks an action of an agonist. Endogenous (such as hormones and neurotransmitters) or exogenous (such as drugs) agonists and antagonists, either stimulate or inhibit a biological response in receptors (Brunton et al. 2008).

## 3. VPython applied to 3D QSAR

How VPython applies to 3D QSAR is basically this: The modelling of molecules and receptors allowing researchers to better tweak the molecular composition and formulation of drug molecules to accurately fit receptors. This is also known as the drug docking problem.

Docking is the fitting of a drug molecule to a receptor. This is a combinatorial problem as there are six degrees of freedom in fitting a ligand to millions of possible sites in a bodily target region.

AutoDock (Scripps Research Institute 2009) is an example of such a docking tool already available. It is beyond the scope of this paper to present an equivalent system; what we will show is how VPython could be applied if we wanted to build a tool of such sophistication.

### 3.1.1 A VPython 3D grid to aid in locating receptors

This is a function to draw 3D grids using the Cylinder object:

```
def Draw3Dgrids():
    length = 20
    rad = 0.01
    lim = 20.0
    step = 2.0

    a = 0.0
    while a <= lim:
        b = 0.0
        while b <= lim:
            cylinder(pos=(a,0,b),axis=(0,length,0),radius=rad,color=color.green)
            cylinder(pos=(0,a,b),axis=(length,0,0),radius=rad,color=color.blue)
            cylinder(pos=(a,b,0),axis=(0,0,length),radius=rad,color=color.white)
            b = b + step
        a = a + step
```

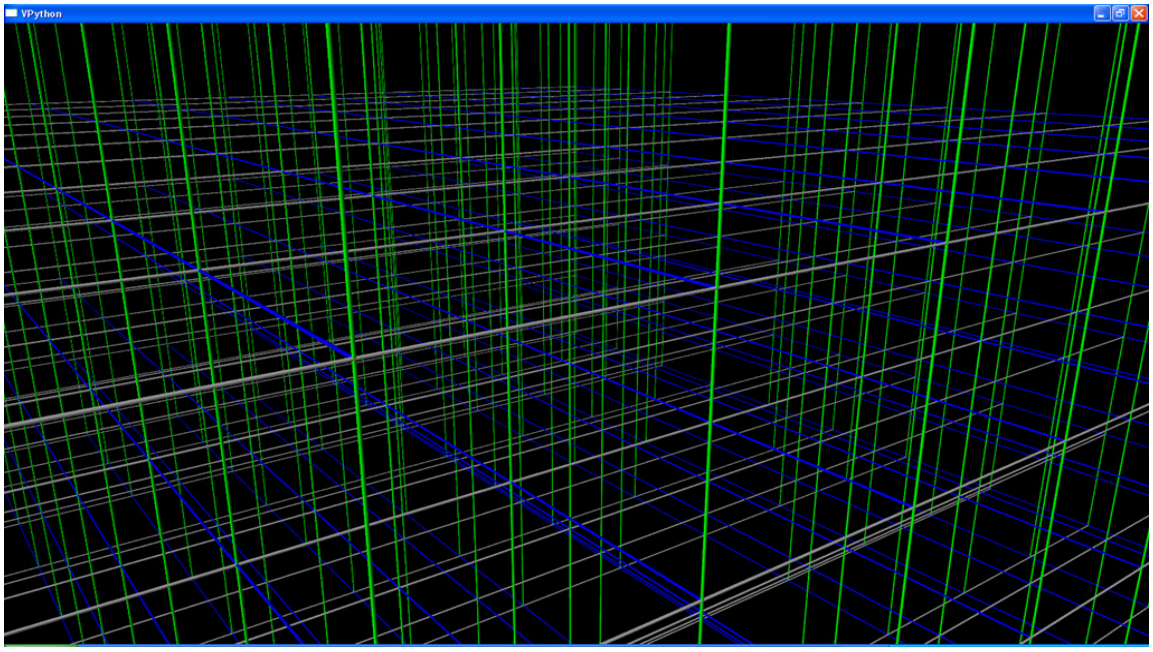



Figure 9. 3D Grids

### 3.1.2 Representation of atoms and molecules in VPython

Let there be a molecule M and a receptor R.

We use the following Python dictionary hash structures to represent atoms:

```
moleculeM = []
atom = []
atom = dict((
  ('type',kTextOxygen),
  ('pos',(5,7,0)),
  ('connect',-1),
  ('dist',0),
  ('size',1.4),
  ('atom',None)
))

moleculeM.append(atom)
```

These atoms are appended to moleculeM list for ease of processing later.
The 'pos' is the position of the atom in a normalised point in 3D space.
The last item 'atom' is a holder for linking the data structure to a VPython instantiation of a sphere.

Assigning a colour to a particular atom:

```
colors = { kTextHydrogen: color.red, kTextCarbon: color.green,
           kTextNitrogen: color.blue, kTextOxygen: color.orange }

for mol in moleculeM:
      atomcolor = colors.get(mol['type'], color.white)
      mol['atom'] = sphere(pos=mol['pos'],radius=mol['size'],
      color=atomcolor)
```

Assigning a VPython sphere to a particular list item:

```
  moleculeM[i]['atom'] = sphere(
      pos=moleculeM[i]['pos'],
      radius=moleculeM[i]['size'],color=atomcolor)
```

### 3.2 Representing receptors

kReceptorTypeMembrane = 'Membrane'

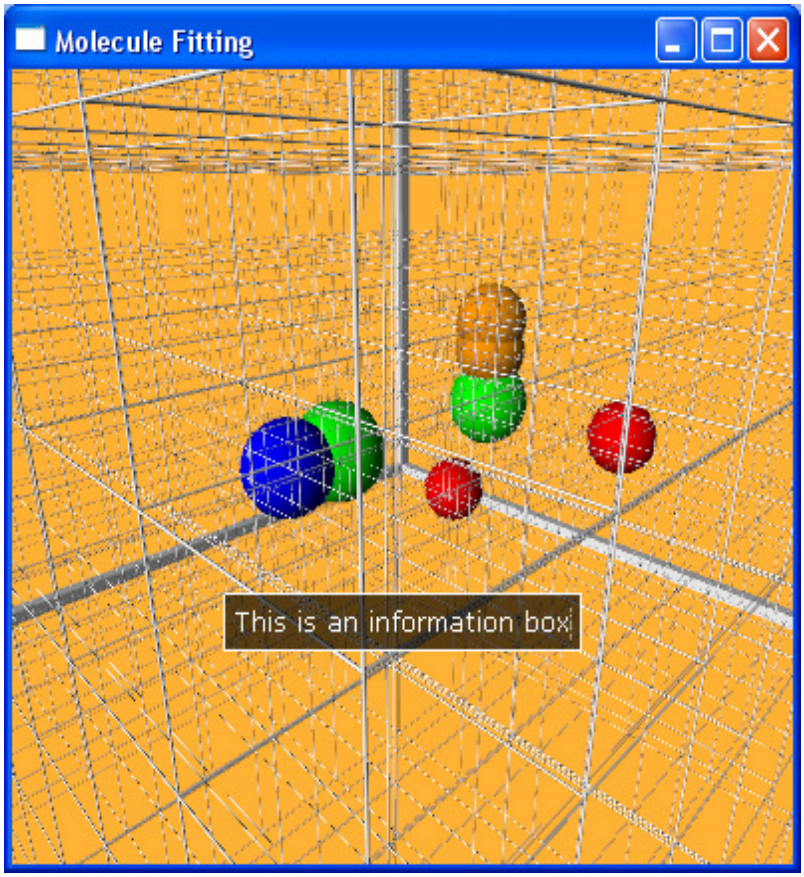
```
kReceptorTypeMetabotropic  = 'Metabotropic'
kReceptorTypeHeptahelical  = 'Heptahelical'
kReceptorTypeGproteinCoupled = 'GproteinCoupled'
kReceptorTypeAdenosine = 'Adenosine'

receptorR = []

site = dict(( ('pos',(x,y,z)),  ('type',kReceptorTypeMembrane)  ))
receptorR.append(site)
...
site = dict(( ('pos',(xn,yn,zn)),  ('type',kReceptorTypeMembrane)  ))
receptorR.append(site)
```

Each site is represented by 'pos' for the position in 3D normalized space, along with the type of receptor.

**The Molecule Fitting Window**




This is where the user adjusts molecule M to fit receptor R. VPython makes a new window via the 'display' command.

```
w = 400
scene = display(x=w, y=0,
width=w, height=w, range=50,
forward=-vector(0,1,1),
newzoom=1)
```

The VPython handle variable 'scene' allows for changes to the display. To change the title and the background colour:

```
scene.title = 'Molecule
Fitting'
scene.background =
(1,0.7,0.2) #copper colour
```

Figure 10. The Molecule Fitting window

The computer mouse left button can do 3 things in this VPython window: 1. Select molecules. 2. Drag molecules around in a fast but imprecise manner. 3. Operate controls for accurate but slow placement of atoms. The right mouse button can rotate the scene in the 3 axes. Both the buttons held down together zooms the scene.

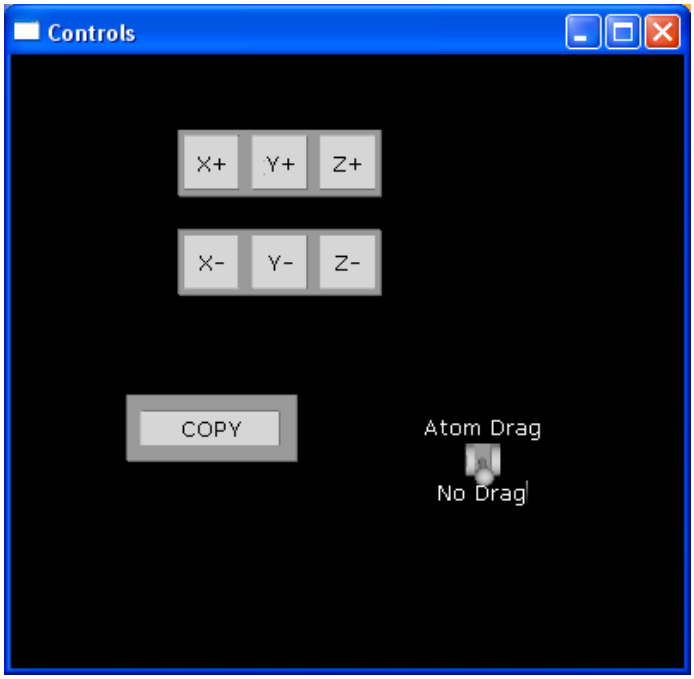



Figure 11. The Controls window

When a molecule is selected, the information box outputs what kind of molecule it is and its position. Fine tuning of its position can be done by the controls window (Figure 11), which is generated by VPython commands.

```
fcb = []     #fcb function callback list
fcb.append(lambda: xchange(+kMovementDistance))
fcb.append(lambda: ychange(+kMovementDistance))
fcb.append(lambda: zchange(+kMovementDistance))
fcb.append(lambda: xchange(-kMovementDistance))
fcb.append(lambda: ychange(-kMovementDistance))
fcb.append(lambda: zchange(-kMovementDistance))
ci = 0
for iy, label2 in enumerate('+-'):
    for ix, label1 in enumerate('XYZ'):
        button(pos=(kbuttonx+ix*20, kbuttony-iy*30), height=20, width=20,
        text=label1+label2,  action=fcb[ci])
        ci = ci + 1

button(pos=(-40,-20),height=20,width=50,text='COPY',action=lambda: atomcopy())

t1 = toggle(pos=(40,-30), width=10, height=10, text1='Atom Drag', text0='No
Drag', action=lambda: ToggleDragControl())
```

All the controls require callbacks to the code via the 'lambda:' parameter.

In Python, 'lambda' or 'nameless functions' are a potential source of confusion for newcomers to the language. However, they are a necessary mechanism in VPython to connect the library with the user's code. We are familiar with the idea of calling a function in a computer language. Lambda functions answer the question: what if the function called needs to call another function during runtime? In the button function call above, the lambda function allows the button to call the user's code after a button is pressed in the graphical user interface. Think of lambda functions as leaving your phone number with an office receptionist when you cannot reach the person you intend to talk to. When the person gets the message, they will call you back.

'Copy' button copies the currently selected atom and places its duplicate near the original.
'Atom Drag' allows for the dragging of atoms via the mouse.

### 3.3 Rules of interaction

In any simulation, there will be a rule database to govern the actions of the elements simulated. For atoms, rules simulated could be the force fields around atoms, distance between atoms, and so on.

#### 3.3.1 Example rule: No Carbon atoms can be within one atom distance from one another.

If there were such a rule, then iterating through moleculeM and pairwise comparison with other atoms using the distance formula for 3D points and checking the 'type' field for 'Carbon':

```
def RuleNoCarbonPairing():
    for i in range(len(moleculeM)):
        if moleculeM[i]['type'] == kTextCarbon:
            for u in range(i+1,len(moleculeM)):
                if moleculeM[u]['type'] == kTextCarbon:
                    p1 = moleculeM[i]['pos']
                    p2 = moleculeM[u]['pos']
                    dist = Distance3D(p1,p2)
                    dist = round(dist,2)
                    if dist < 4:
                        print kTextCarbon,' atoms ',str(i),' and ',
```

```
                    str(u),' too close'
```

Distance between 2 points in 3D formula and code:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

```
def Distance3D(p1,p2):
    Ax,Ay,Az = p1
    Bx,By,Bz = p2
    dx = Ax-Bx
    dy = Ay-By
    dz = Az-Bz
    distance = sqrt(dx*dx + dy*dy + dz*dz)
    return distance
```

### 3.3.2 Determining the fit to the receptor

Again, iterating through a selected atom and finding the nearest neighbour in the receptorR list:

```
def FindClosestReceptor():
    print 'FindClosestReceptor'
    for i in range(len(receptorR)):
        p1 = receptorR[i]['pos']
        p2 = currAtom['pos']
        dist = Distance3D(p1,p2)
        dist = round(dist,2)
        print 'dist',dist,'receptor ',i,'at',p1
```

Outputs show that the atom selected is closest to receptor 4 in this example run:

```
FindClosestReceptor
dist 5.59 receptor  0 at (5, 7, 5)
dist 7.43 receptor  1 at (5, 7, 3)
dist 9.34 receptor  2 at (5, 7, 1)
dist 3.91 receptor  3 at (5, 7, 7)
dist 2.69 receptor  4 at (5, 7, 9)
FindClosestReceptor
dist 5.39 receptor  0 at (5, 7, 5)
dist 7.28 receptor  1 at (5, 7, 3)
dist 9.22 receptor  2 at (5, 7, 1)
dist 3.61 receptor  3 at (5, 7, 7)
dist 2.24 receptor  4 at (5, 7, 9)
FindClosestReceptor
dist 5.22 receptor  0 at (5, 7, 5)
dist 7.16 receptor  1 at (5, 7, 3)
dist 9.12 receptor  2 at (5, 7, 1)
dist 3.35 receptor  3 at (5, 7, 7)
dist 1.8 receptor  4 at (5, 7, 9)
FindClosestReceptor
dist 5.1 receptor  0 at (5, 7, 5)
dist 7.07 receptor  1 at (5, 7, 3)
dist 9.06 receptor  2 at (5, 7, 1)
dist 3.16 receptor  3 at (5, 7, 7)
dist 1.41 receptor  4 at (5, 7, 9)
FindClosestReceptor
dist 5.02 receptor  0 at (5, 7, 5)
```

```
dist 7.02 receptor  1 at (5, 7, 3)
dist 9.01 receptor  2 at (5, 7, 1)
dist 3.04 receptor  3 at (5, 7, 7)
dist 1.12 receptor  4 at (5, 7, 9)
FindClosestReceptor
dist 5.0 receptor  0 at (5, 7, 5)
dist 7.0 receptor  1 at (5, 7, 3)
dist 9.0 receptor  2 at (5, 7, 1)
dist 3.0 receptor  3 at (5, 7, 7)
dist 1.0 receptor  4 at (5, 7, 9)
FindClosestReceptor
dist 4.5 receptor  0 at (5, 7, 5)
dist 6.5 receptor  1 at (5, 7, 3)
dist 8.5 receptor  2 at (5, 7, 1)
dist 2.5 receptor  3 at (5, 7, 7)
dist 0.5 receptor  4 at (5, 7, 9)
FindClosestReceptor
dist 4.0 receptor  0 at (5, 7, 5)
dist 6.0 receptor  1 at (5, 7, 3)
dist 8.0 receptor  2 at (5, 7, 1)
dist 2.0 receptor  3 at (5, 7, 7)
dist 0.0 receptor  4 at (5, 7, 9)
```

### 4.0 Conclusion

The goal of this paper is to demonstrate the main features of VPython, and not to make a complete molecular docking system. For any kind of simulation tool, the usefulness is determined by the accuracy of the input data, the rule base, and the link between the interface the the underlying data.VPython makes the demonstration of objects in a 3D space over time a relatively easy task. It brings physics data and formulae to life in a manner not easily matched by other media.

Molecular fitting is only half the story in drug design. While a drug can be shown to fit well within a receptor in silico, this does not imply that good biological activity in vitro or in vivo is guaranteed. This is because biological activity of a drug is dependent on numerous factors which include its pharmacokinetics (absorption, distribution, metabolism and excretion). Research is still undergoing to reliably incorporate such predictions in computer drug design systems.

In other words, eventually the drug designed in silico must still be synthesised for testing of its biological activity. Although this is true, QSAR is an important technique which can help reduce the cost and shorten the drug design process by quickly focusing on compounds with good predicted biological activities. VPython can be made used to perform the docking functions of QSAR.

## References

Bevan, D. (1997). *QSAR and Drug Design* [Online] (Updated March 1997) Available at: http://www.biochem.vt.edu/modeling/qsar_drug.html [Accessed 11 January 2010].

Brown, T., LeMay, H. E., Bursten, B. E., Murphy, C. J., & Woodward, P. (2009), *Chemistry: The Central Science, AP Edition (11th ed.)*, Upper Saddle River, NJ: Pearson/Prentice Hall, pp. 5–6.

Brunton, L., Blumenthal, D., Buxton, I., Parker, K., *Goodman and Gilman's Manual of Pharmacology and Therapeutics*. (11th edition, 2008). p14.

Dudek A., Arodzb T., & Gálvez J. (2006). Computational Methods in Developing Quantitative

Structure-Activity Relationships (QSAR): A Review, *Combinatorial Chemistry & High Throughput Screening*, 2006, 9, 213-228

Farlax, Inc., (2010). *Encyclopedia article about Medicinal Chemistry* [Online] (Updated 2010?) Available at: http://encyclopedia.thefreedictionary.com/Medicinal+chemistry [Accessed 20 January 2010].

Fujita, T., Iwasa, J., & Hansch, C. (1964). *Journal of the American Chemical Society*. Volume 86, pp 5175–5180.

Hammett, L. P. (1940). *Physical Organic Chemistry, McGraw-Hill Book Co., Inc.*, New York, NY, 1940, Chaps. III,IV,VII.

Salgado, R. (2009). *Rob's page of VPython applications for Teaching Physics* [Online] (Updated 6 August 2009) Available at: http://www.visualrelativity.com/vpython/#realtimedata2009 [Accessed 10 January 2010].

Scherer, D., Dubois, P., & Sherwood, B. (2000). VPython: 3D Interactive Scientific Graphics for Students*, Computing in Science and Engineering,* Sept./Oct. 2000, 82-88.

Scherer, D., (2009). *3D Objects* [Online] (Updated 23 December 2009) Available at: http://vpython.org/contents/docs/visual/primitives.html [Accessed 7 January 2010].

Scripps Research Institute (2009). *AutoDock* [Online] (Updated 8 October 2009) Available at: http://autodock.scripps.edu/ [Accessed 21 January 2010].

Selassie C.D. (2003) *Burger's Medicinal Chemistry and Drug Discovery Sixth Edition*, Volume 1: Drug Discovery Edited by Donald J. Abraham. John Wiley&Sons, Inc. pp. 1-7.

Siegel, A. (2006). *PyGeo: A dynamic geometry toolkit* [Online] (Updated January 2006) Available at: http://pygeo.sourceforge.net/ [Accessed 10 January 2010].

Sprous, D. G., Zhang J., Zhang L., Wang Z., & Tepper M. A., (2006) Kinase inhibitor recognition by use of a multivariable QSAR model, *Journal of Molecular Graphics and Modelling*, Volume 24, Issue 4, January 2006, pp. 278-295.

Thompson, E. (2009). *VPython - Physics and 3D in Python* [Online] (Updated 29 June 2008) Available at: http://showmedo.com/videos/series?name=pythonThompsonVPythonSeries [Accessed 10 January 2010].

Urbano, L. (2005) Water Phase Changes [Online] (Updated January 20th, 2005) Available at: http://des.memphis.edu/lurbano/vpython/phase/phase.html [Accessed 10 January 2010].